Programming Principles

Unit 6

Functions

Some functions

that you may have used already

Some Existing Functions

fire(…);

move(…);

grab();

wait(…)

turn(…);

strval(…)

sqrt(…)

strupper(…)

jet(…)

drop()

pendown()

aim(…)

pow(…, …)

Most instructions you have used are **functions!**

What is a function?

Does something useful

Has a unique name

Can be used in many different programs

Is 'called' by using its name

Has brackets after its name

Can be used many times in the same program

May use parameters (values passed in through the brackets)

May also return a value back

Functions with no parameters

grab();

picks up object in front

drop();

drops object

pendown();

puts pen down onto floor

green();

changes colour to green

Functions with parameters

```
move(…);
move( 30 );
```

```
turn(…);
turn( 120 );
```

```
wait(…);
wait( 2.5 );
```

parameters passed into function

Functions that return values

pow(…)

cube = **pow (number, 3) ;**

value returned

strupper(…)

choice = **strupper (choice);**

value returned

sqrt(…)

value = **sqrt (number) ;**

value returned

parameters passed into function

User-defined functions

Designing your own functions

Creating a drawCircle() function

Defining the drawCircle() function

function name

void here means no value is being returned

no parameters are being used

```
void object:: drawCircle ()
{
    pendown();
    for (int i = 0; i < 36; i++)
    {
        move(0.1) ;
        turn(10) ;
    }
    penup();
}
```

Every function has { and }

This code is executed when the function is called in a program

function call

```
drawCircle();
```

Using drawCircle()

```
extern void object::DrawIt()
{
    blue();
    drawCircle();     // call drawCircle()
    move(5);
    yellow();
    drawCircle();     // call drawCircle() again
}
```

Program execution starts here

This main program calls drawCircle() twice to draw 2 circles

```
void object:: drawCircle ()
{
    pendown();
    for (int i = 0; i < 36; i++)
    {
        move(0.1) ;
        turn(10) ;
    }
    penup();
}
```

The **drawCircle()** code runs only when the function is called

Function definitions are put beneath the main program

Functions and loops

Repeating a function call

Drawing 10 circles

```
extern void object::Draw10Times()
{
    blue();
    for (int i = 0; i < 10; i++)
    {
        drawCircle();
        move(5);
    }
}
```

Program execution starts here

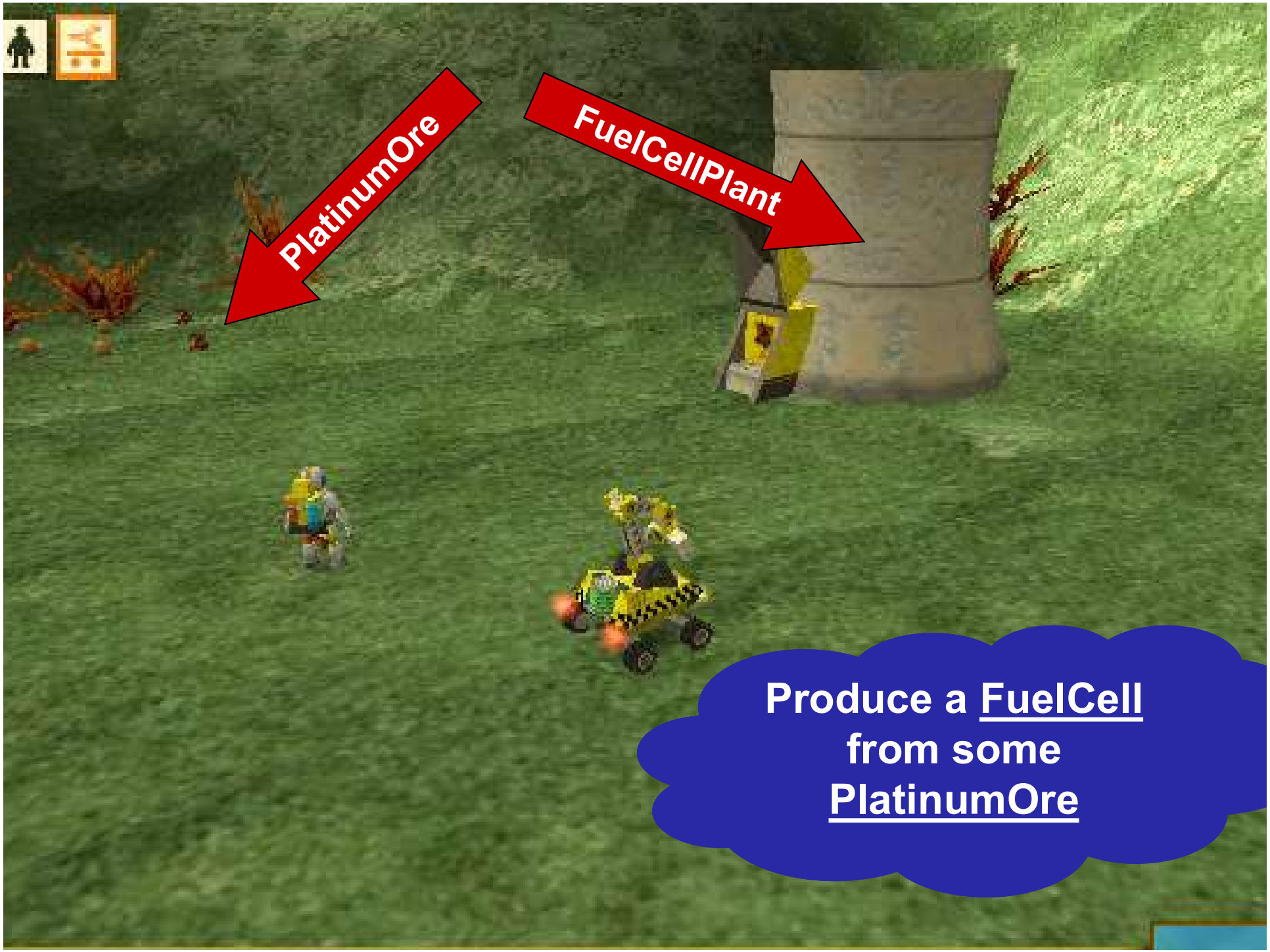
The main program calls drawCircle() 10 times

```
void object:: drawCircle ()
{
    pendown();
    for (int i = 0; i < 36; i++)
    {
        move(0.1) ;
        turn(10) ;
    }
    penup();
}
```

The **drawCircle()** code is called 10 times but only written once

Using functions in larger programs

Divide and Rule!


PlatinumOre
FuelCellPlant
Produce a FuelCell
from some
PlatinumOre

What are the steps?

1. Find and pick up some PlatinumOre

2. Use FuelCellPlant to create FuelCell

We shall use 2 functions
to perform these steps

getOre() function

This should find PlatinumOre, go to it and pick it up

```
void object::getOre()
{
    object item;

    item = radar(PlatinumOre);
    goto(item.position);
    grab();
}
```

convertOre() function

Drop the ore at the FuelCellPlant, wait, then pick up the FuelCell

```
void object::convertOre()
{
    object item;

    item = radar(FuelCellPlant);
    goto(item.position);
    drop();
    move(-3);  // get out of way
    wait(12);
    move(3);
    grab();
}
```

better

```
while (radar(FuelCell) == null)
{
        wait(0.1);
}
```

Getting it all to work()

```
extern void object::functionProg()
{
    getOre();
    convertOre();
}
```

Each function is called in turn in the main program

```
void object::getOre()
{
    object item;
    item = radar(PlatinumOre);
    goto(item.position);
    grab();
}
```

```
void object::convertOre()
{
    object item;
    item = radar(FuelCellPlant);
    goto(item.position);
    drop();
```

functions are defined below the main program

Repeating It

There are

2 pieces of ore

Repeat it twice

```
extern void object::functionRepeats()
{
    int  count;
    for (count=0; count<2; count++)
    {
        getOre();
        convertOre();
    }
}
```

Repeat the calls to getOre() and convertOre()

```
void object::getOre()
{
    object item;
    item = radar(PlatinumOre);
    goto(item.position);
    grab();
}
```

What is wrong?

We need another function to remove the FuelCell and drop it somewhere

```
void object::convertOre()
{
    object item;
```

Repeat for lots of Ore

```
extern void object::functionRepeatsaLot()
{
    while ( radar(PlatinumOre) != null )
    {
        getOre();
        convertOre();
    }
}
```

Repeat as long as there is PlatinumOre available

```
void object::getOre()
{
    object item;
    item = radar(PlatinumOre);
    goto(item.position);
    grab();
}
```

We still need a function to remove the FuelCell and drop it somewhere

```
void object::convertOre()
{
    object item;
```

Local variables

Example of local variables

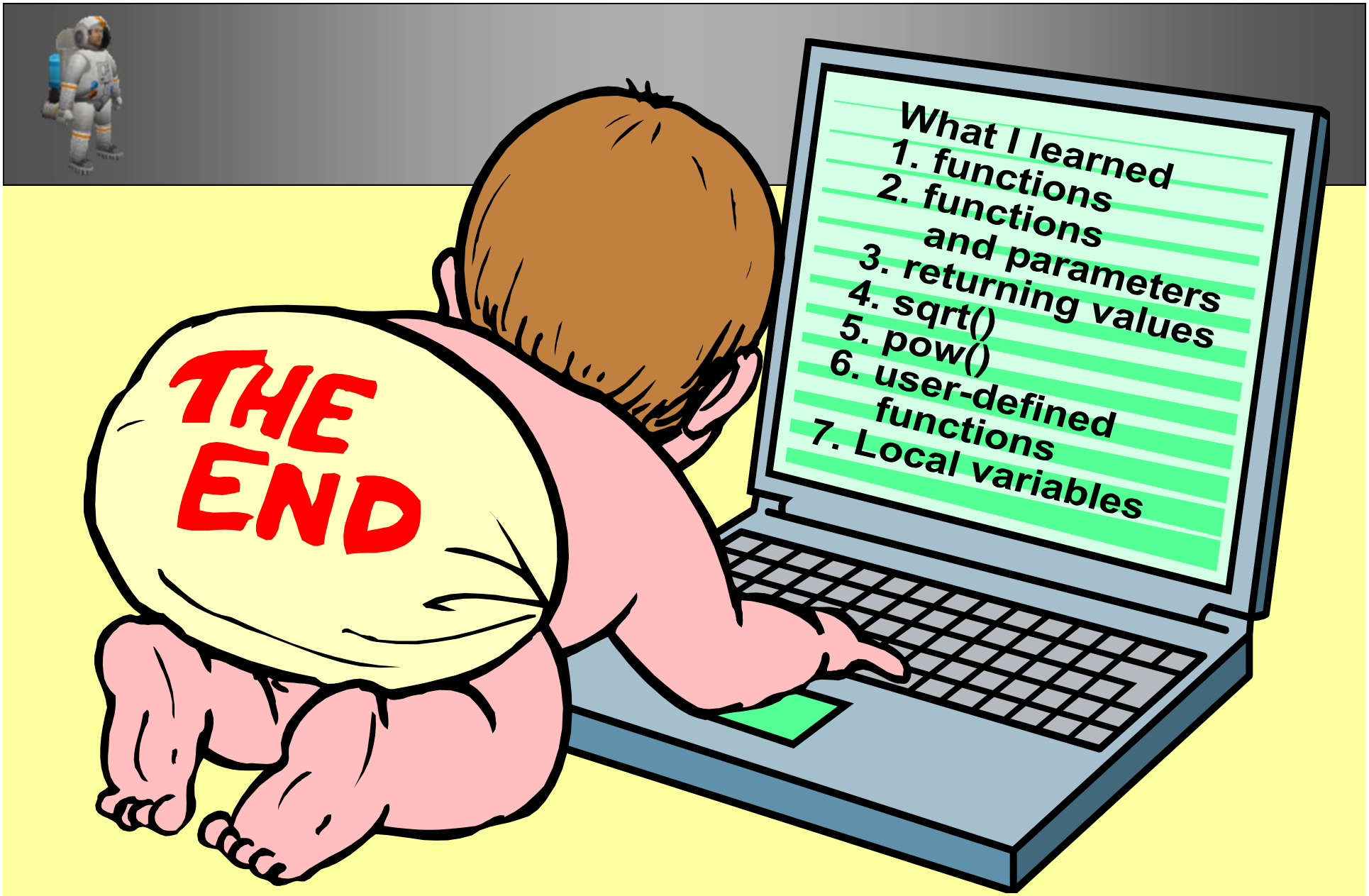
```
extern void object::functionProg()
{
    getOre();
    convertOre();
}
```

```
void object::getOre()
{
    object  item;
    item = radar(PlatinumOre);
    goto(item.position);
    grab();
}
```

```
void object::convertO()
{
    object  item;
    item = radar(FuelCellPlant);
    goto(item.position);
    drop();
```

item is here a **local** variable for each function

these are **different** variables .. even though they have the same name!

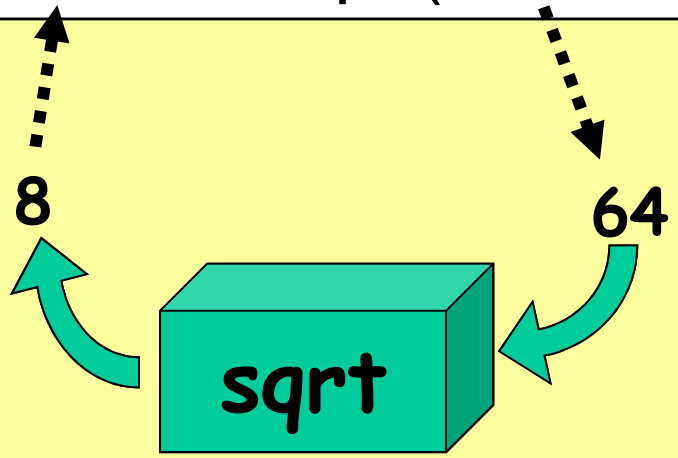

What I learned
1. functions
2. functions and parameters
3. returning values
4. sqrt()
5. pow()
6. user-defined functions
7. Local variables
THE END

Extra Reading

How the sqrt() function Works

```
input = dialog( "Enter a number ") ;
number = strval(input);
```

```
value = sqrt ( number );
```




```
message( "The square root is " + value) ;
```

Local Variables

- These are declared **inside** a function
  - -- and can only be used in that function
  - -- they are **not** recognised outside the function
- Local variables are **created** when the function is called
  - -- and **destroyed** when the function finishes
- They help to make functions more **independent**
  - -- so they can be used in other programs without messing them up
- We say that the **scope** of the variable is the function in which it is declared

Why use functions?

Why use functions?

- Large programs can be broken up into smaller sections
- Programs are then easier to understand
- It is easier to modify programs
- It is easier to locate errors
- division of work among programming teams is easier
- functions can be re-used in other programs
- saves duplicating code (write once .. use many times)
- creates better program structure
- makes programs more:
  - readable
  - maintainable
  - reliable
  - and less complex